TÉCNICAS DE PROGRAMACIÓN

ORDENAMIENTO Y BÚSQUEDA

FUNCIONES ANÓNIMAS

```
add = lambda x, y: x + y
print(add(3, 5)) # 8
```

anónimas

MAP

MAP APLICA UNA FUNCIÓN A TODOS LOS ELEMENTOS DE UNA LISTA

```
items = [1, 2, 3, 4, 5, 6]

def multiplay_elements_by_two(numbers):
    result = []
    for number in numbers:
        result.append(number * 2)
    return result

result = multiplay_elements_by_two(items)


result = list(map(lambda x: x * 2, items))
```

FILTER

FILTER CREA UNA LISTA DE ELEMENTOS PARA LOS QUE UNA FUNCIÓN DEVUELVE TRUE

```
items = [1, 2, 3, 4, 5, 6]

def get_low_than_three(numbers):
    result = []
    for number in numbers:
        if number < 3:
            result.append(number)
    return result
result = get_low_than_three(items)


result = list(filter(lambda x: x < 3, items))
```

REDUCE

REDUCE ES UNA FUNCIÓN APLICA UN CÁLCULO EN UNA LISTA Y DEVOLVIENDO UN ÚNICO RESULTADO.

```
items = [1, 2, 3, 4, 5, 6]
```

```
def get_average(numbers):
    result = 0
    for number in numbers:
        result += number
    return result / len(numbers)

average = get_average(items)
```

```
from functools import reduce
average = reduce((lambda x, y: x * y), items) / len(items)
```

REDUCE

REDUCE ES UNA FUNCIÓN APLICA UN CÁLCULO EN UNA LISTA Y DEVOLVIENDO UN ÚNICO RESULTADO.

```
items = [1, 2, 3, 4, 5, 6]
```

```
def get_average(numbers):
    result = 0
    for number in numbers:
        result += number
    return result / len(numbers)

average = get_average(items)
```

```
from functools import reduce
average = reduce((lambda x, y: x * y), items) / len(items)
```

USALOS CON CUIDADO

```
findShort = (s) => Math.min.apply(null, s.split(' ').map( (i) => i.length ))
```

```
function findShort (sentece) {
  words = sentece.split(' ')
  minWordSize = words[0].length

  words.forEach( word => {
    wordSize = word.length
    minWordSize = wordSize < minWordSize ? wordSize : minWordSize
  })

  return minWordSize
}
```

ALGORITMOS DE ORDENAMIENTO

SECTION SORT

1. DIVIDE LA INFORMACIÓN EN ORDENADO Y NO ORDENADO
2. ENCUENTRA EL ELEMENTO MÍNIMO DE LA SECCIÓN NO ORDENADA
3. AGRÉGALO A LA SECCIÓN ORDENADA


```python
def section_sort(numbers):
    result = numbers.copy()
    for i in range(len(result)):
        min_index = i
        for j in range(i + 1, len(result)):
            if result[min_index] > result[j]:
                min_index = j
        result[i], result[min_index] = result[min_index], result[i]
    return result
```

O(N^2)

BUBBLE SORT

1. INICIA EL INDICE I EN 0
2. COMPARARA EL ELEMENTO [I] E [I + 1]
3. SI EL ELEMENTO [I + 1] > [I] INTERCAMBIAMOS DE LUGAR
4. REPITES CUANTAS VECES SEA NECESARIO

5 2 4 6 1 3

```python
def bubble_sort(numbers):
    result = numbers.copy()
    size = len(numbers)

    for i in range(size):
        for j in range(0, size - i - 1):
            if result[j] > result[j + 1] :
                result[j], result[j + 1] = result[j + 1], result[j]
    return result
```

O(N^2)

INSERTION SORT

1. INICIA EL INDICE I EN 0
2. COMPARA CON EL SIGUIENTE ELEMENTO
3. SI ES MAYO INCREMENTA EL INDICE
4. SI ES MENOR TOMA EL NUMERO MENOR Y ENCUENTRA SU LUGAR EN UN INDICE ANTERIOR

6 5 3 1 8 7 2 4

```python
def insertion_sort(numbers):
    result = numbers.copy()
    for i in range(1, len(result)):
        key = result[i]
        j = i - 1
        while j >= 0 and key < result[j] :
                result[j+1] = result[j]
                j -= 1
        result[j + 1] = key
    return result
```

O(N^2)

MERGE SORT

1. DIVIDE EL ARREGLO EN DOS
2. SE PUEDE SEGUIR DIVIDIENDO EN 2, DIVIDE
3. JUSTA Y ORDENA

6 5 3 1 8 7 2 4

MERGE SORT

```python
def merge(left, right):
    left_size = len(left)
    right_size = len(right)
    if not left_size or not right_size:
        return left or right

    result = []
    i, j = 0, 0
    while (len(result) < left_size + right_size):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
        if i == left_size or j == right_size:
            result.extend(left[i:] or right[j:])
            break
    return result
```

```python
def merge_sort(numbers):
    if len(numbers) < 2:
        return numbers
    middle = len(numbers) // 2
    left = merge_sort(numbers[:middle])
    right = merge_sort(numbers[middle:])
    return merge(left, right)
```

O(N LOGN)

QUICK SORT

1. ELEGIR UN ELEMENTO ALEATORIO DEL MEDIO DEL ARREGLO
2. RESITUA LOS DEMÁS ELEMENTOS DE LA LISTA A CADA LADO DEL PIVOTE MENORES A LA IZQUIERDA, MAYORES A LA DERECHA
3. REPITE CON LAS DOS NUEVAS LISTA IZQUIERDA Y DERECHA

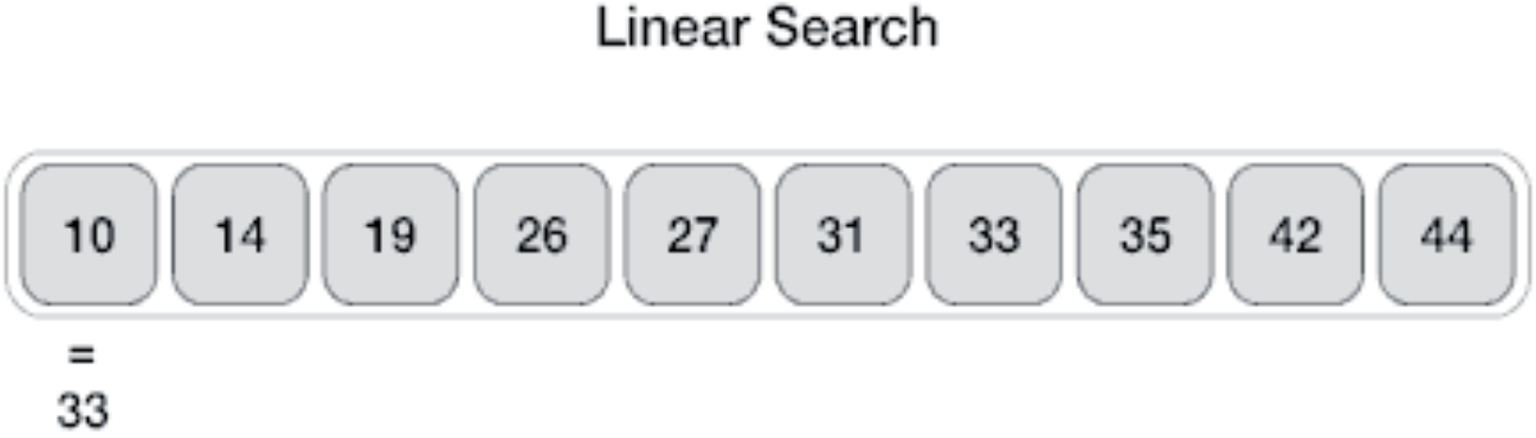
QUICK SORT

```python
def quick_sort(numbers):
    size = len(numbers)
    if size == 1:
        return numbers

    middle = size // 2
    pivot = numbers[middle]
    low_numbers = []
    high_numbers = []

    for index, number in enumerate(numbers):
        if index == middle:
            continue
        if number > pivot:
            high_numbers.append(number)
        else:
            low_numbers.append(number)
    return quick_sort(low_numbers) + [pivot] + quick_sort(high_numbers)
```

O(N LOGN)

ALGORITMOS DE BÚSQUEDA

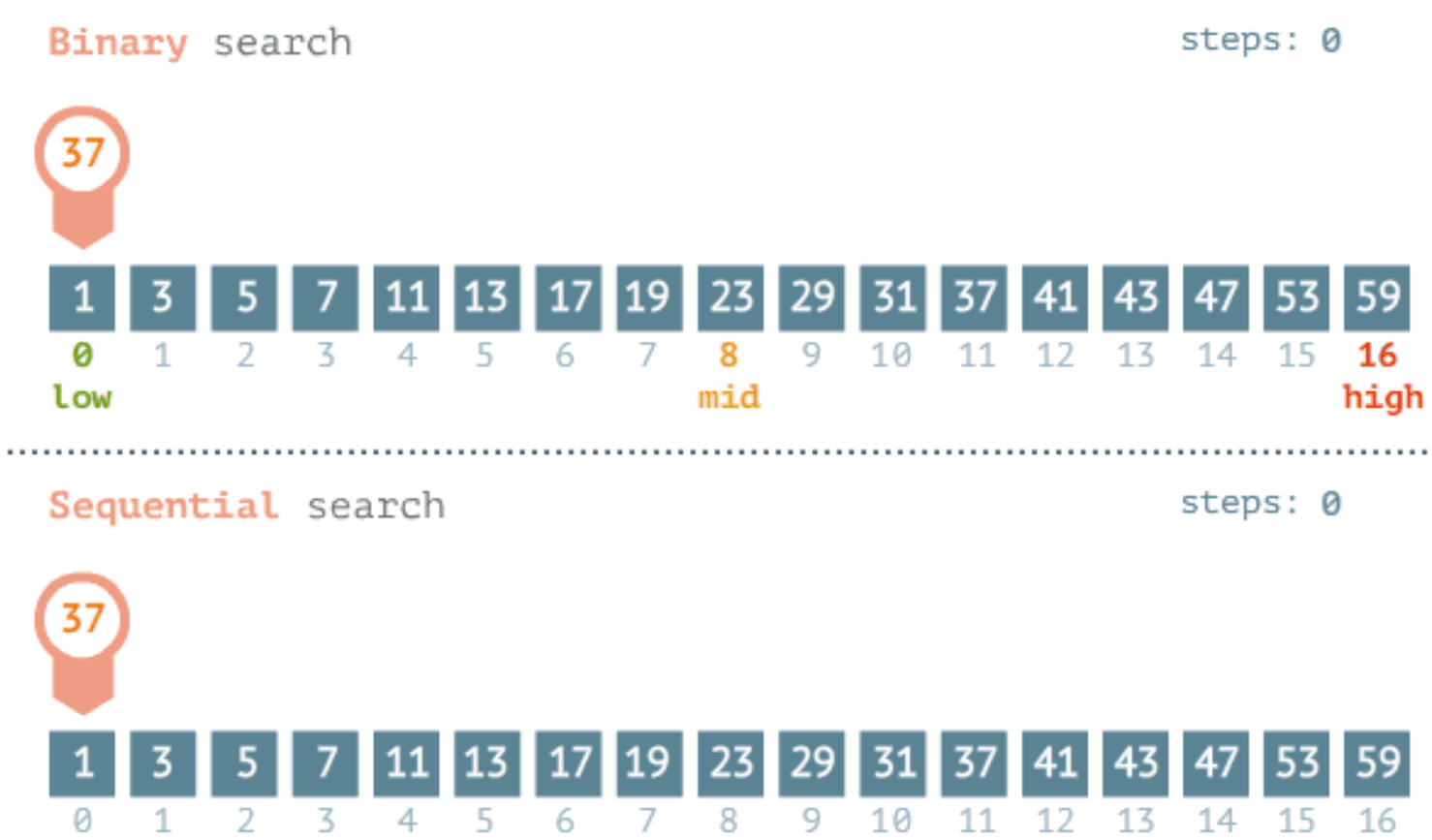
BÚSQUEDA LINEAL




```python
def search(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1
```

O(N)

BÚSQUEDA BINARIA



www.penjee.com

1. ALGORITMOS
2. BIG O NOTATION
3. GRÁFICAS DE BIG O
4. COMPLEJIDAD DE OPERACIONES EN PYTHON
5. METODOLOGIAS PARA RESOLVER ALGORITMOS
6. RECURSIVIDAD
7. MEMORIZACIÓN